Санкт-Петербургский государственный университет

Кафедра компьютерного моделирования и многопроцессорных систем

Сатина Алена Юрьевна

Выпускная квалификационная работа бакалавра

Разработка инструментария для визуализации сильно нелинейных процессов на основе свободно распространяемого программного обеспечения

Направление 010400

«Прикладная математика и информатика»

Научный руководитель,

доктор техн. наук,

профессор

Дегтярев А. Б.

Санкт-Петербург

2017

Содержание

Введение

Нелинейные процессы представляют собой один из основных и наиболее распространенных в природе и технике видов явлений. Колебания маятника, электромагнитные и звуковые волны, волны на поверхности воды образуют лишь часть тех нелинейных явлений, которые широко встречаются вокруг нас. Постоянное развитие информационных технологий дает возможности для визуализации явлений и процессов нашего мира, с которыми мы сталкиваемся на каждом шагу, как в повседневной жизни, так и при более глубоком изучении физики явлений.

С математической точки зрения нелинейные явления описываются соответствующими дифференциальными уравнениями, которые не всегда удается быстро решить.

Задачи решения таких уравнений очень актуальны в наше время, но на стандартных архитектурах считаются медленно, в связи с невозможностью массовых расчетов.

Для увеличения скорости решения прикладных задач, с целью их эффективного исполнения, широко используется применение идей распараллеливания алгоритмов. В настоящее время существует множество вычислительных систем, которые используют различные принципы параллельной обработки данных.

Одним из перспективных направлений в развитии таких систем являются гибридные комплексы и гетерогенные системы. Для вычислительных целей используется графический процессор GPU, что является основным направлением в данной сфере.

Разработано множество различных архитектур параллельных вычислений, таких как: CUDA, OpenCL и др., позволяющих повысить производительность вычислений. Но для использования подобных технологий необходимо иметь глубокие знания об устройстве данных продуктов, а для их изучения потребуется достаточное количество времени.

К тому же, иногда перенос на гибридные архитектуры затруднен, в связи с трудностью управления потоками внутри GPU.

Помимо данных платформ существуют библиотеки и открытые стандарты параллельных алгоритмов, позволяющие строить программы и не требующие определенных навыков и знаний об устройстве технологий, реализующих параллельную обработку данных.

Постановка задачи

Целью данной работы является разработка и распараллеливание алгоритма численного решения дифференциальных уравнений, которые описывают нелинейные волновые процессы. Задача выполняется на примере уравнения Кадомцева-Петвиашвили. Требуется решить задачу разработки параллельной программы, основанной на методе пятиточечной прогонки.

Для достижения данной цели необходимо проанализировать предметную область, изучить основные сведения о дифференциальных уравнениях, описывающих нелинейные волновые процессы.

Кроме того, требуется исследовать архитектуры, которые можно использовать для распараллеливания алгоритма.

Обзор литературы

Для выполнения данной работы необходимо изучить две основные предметные области.

Первая включает в себя базовые основы теории нелинейных процессов и применение различных методов при решении дифференциальных уравнений, описывающих такие процессы.

Вторая предметная область охватывает спектр знаний об основных принципах работы с архитектурами параллельных вычислений.

Книга Флетчера “Вычислительные методы для гидродинамики” [6] содержит сведения о конкретных техниках и основах для развития навыков при использовании методов в различных отраслях вычислительной гидродинамики.

В данной книге проиллюстрированы основные вычислительные методы, дифференциальные уравнения в частных производных, сходимости, стабильности, последовательности и методы решения уравнений.

Кроме того, здесь содержатся сведения о вычислении конечных разностей, элементов и объемов.

Книга изобилует реальными примерами и решениями различными сложных проблем, возникающих в области гидродинамики.

Для изучения сведений о распараллеливании алгоритмов была проанализирована книга Антонова “Параллельное программирование с использованием технологии OpenMp” [7].

Книга содержит основные понятия о компиляции программы, директивах и функциях, применяемых для распараллеливания задач. Приводится достаточное количество примеров программ, с помощью которых можно понять принципы параллельного программирования. Программы написаны на языке Си и Фортран.

Также немаловажной частью книги являются вопросы и задания, с помощью которых процесс обучения становится более понятным.

Отдельный раздел посвящен использованию OpenMP. Здесь описываются принцип инкрементального программирования, достоинства и недостатки использования данного открытого стандарта, и его совместное применение с другими технологиями.

Глава 1. Теоретические сведения

1.1 Метод пятиточечной прогонки

При нахождении решения для уравнений высоких порядков можно использовать два способа метода пятиточечной прогонки. Первый способ состоит в переходе к системе дифференциальных уравнений первого порядка и построении соответствующей разностной схемы. Второй способ заключается в непосредственной аппроксимации исходной дифференциальной задачи. В этом случае мы приходим к многоточечным разностным уравнениям. Наиболее часто встречаются системы пятиточечных уравнений следующего вида:

$$c_0 y_0 - d_0 y_1 + e_0 y_2 = f_0, \qquad i = 0,$$

$$-b_1 y_0 + c_1 y_1 - d_1 y_2 + e_1 y_3 = f_1, \qquad i = 1,$$

$$a_i y_{i-2} - b_i y_{i-1} + c_i y_i - d_i y_{i+1} + e_i y_{i+2} = f_i, \qquad 2 \le i \le N-2,$$

$$a_{N-1} y_{N-3} - b_{N-1} y_{N-2} + c_{N-1} y_{N-1} - d_{N-1} y_N = f_{N-1}, \quad i = N-1,$$

$$a_N y_{N-2} - b_N y_{N-1} + c_N y_N = f_N, \qquad i = N.$$

Алгоритм прогонки для системы, приведенной выше, запишем в следующем виде:

1. По формулам:

$$\alpha_{i+1} = \frac{1}{\Delta_i}[d_i + \beta_i(a_i\alpha_{i-1} - b_i)], \qquad i = 2, 3, \dots, N-1$$

$$\alpha_1 = \frac{d_0}{c_0}, \alpha_2 = \frac{1}{\Delta_1}(d_1 - \beta_1 b_1),$$

$$\gamma_{i+1} = \frac{1}{\Delta_i}[f_i - a_i\gamma_{i-1} - \gamma_i(a_i\alpha_{i-1} - b_i)], \quad i = 2, 3, \dots, N,$$

$$\gamma_1 = \frac{f_0}{c_0}, \gamma_2 = \frac{1}{\Delta_1}(f_i - \gamma_1 b_1),$$

$$\beta_{i+1} = \frac{e_i}{\Delta_i}, \quad i = 1, 2, \dots, N-2, \qquad \beta_1 = \frac{e_0}{c_0},$$

где $\Delta_i = c_i - a_i\beta_{i-1} + \alpha_i(a_i\alpha_{i-1} - b_i),\ 2 \le i \le N,\ \Delta_1 = c_1 - b_1\alpha_1,$

находятся прогоночные коэффициенты $\alpha_i, \beta_i,\ \gamma_i$;

2. Неизвестные y_i находятся последовательно по формулам:

$$y_i = \alpha_{i+1}y_{i+1} - \beta_{i+1}y_{i+2} + \gamma_{i+1},\ \ i = N-2, N-3, \dots, 0,$$

$$y_{N-1} = \alpha_N y_N + \gamma_N,\ \ y_N = \gamma_{N+1}.$$

Построенный алгоритм будем также называть алгоритмом монотонной прогонки [10].

1.2 Уравнение КдВБ

Рассмотрим уравнение Кортевега - де Вриза – Бюргерса (КдВБ), являющееся универсальным для описания одномерных волн в средах с диссипацией и дисперсией:

$$v_t + vv_x + \alpha v_{xx} + \beta v_{xxx} = \gamma I(v), \quad (1)$$

где α является мерой диссипации, β является мерой дисперсии, γ является мерой межфазного взаимодействия, $I(v)$ является интегральным оператором [1].

При численном интегрировании уравнения КдВБ вместо исходного уравнения (1) рассматривается его аналог, записанный в дивергентной форме:

$$u_t + 0.5(u^2)_x + \alpha u_{xx} + \beta u_{xxx} = \gamma I(v), \quad (2)$$

Решение данного уравнения (2) в полуплоскости $t \geq 0$ ищется для начального распределения $u(x,0) = f(x),\ x \in (-\infty, +\infty)$. Граничные условия – условия Дирихле: $u(-\infty, t) = a,\ u(+\infty, t) = b$.

В основном, численное моделирование уравнения (2) проводится с помощью двухступенчатой явной конечно-разностной схемы Мак-Кормака [5][6] с использованием в некоторых случаях процедуры коррекции потоков [3]. Также есть и другие численные подходы, в частности, обобщение Залесака для гиперболических систем [4].

Непрерывная область для (2) заменяется расчётной областью $[x_{\min},\ x_{\max}] \times [0, T]$. В расчётной области задается равномерная разностная сетка по времени t и по пространственной координате x:

$$u_j^n = u(x_j, t^n),$$

$$x_j = j\Delta x, \qquad j \in [1, M], \qquad x_{\min} = x_1, \qquad x_{\max} = x_M,$$

$$t^n = n\Delta t, \qquad n = 0,1,2, \dots, T/\Delta t - 1.$$

где Δx— шаг пространственной координаты, Δt — шаг по времени. Граничные условия:

$$u(x_{\min}, t) = a, \;\; u(x_{\max}, t) = b.$$

Разностная схема Мак-Кормака для уравнения (2) имеет следующий вид:

Шаг 1: Предиктор.

$$u_j^* = u_j^n - \frac{\Delta t}{2\Delta x}\left[\left(u_{j+1}^n\right)^2 - \left(u_j^n\right)^2\right] - r\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) - s\left(u_{j+2}^n - 2u_{j+1}^n + 2u_{j-1}^n + u_{j-2}^n\right) + \gamma\Delta t I_j^n$$

Шаг 2. Корректор.

$$\bar{u} = \frac{u_j^n + u_j^*}{2} - \frac{\Delta t}{4\Delta x}\left[\left(u_j^*\right)^2 - \left(u_{j-1}^*\right)^2\right] - \frac{r}{2}\left(u_{j+1}^* - 2u_j^* + u_{j-1}^*\right) - \frac{s}{2}\left(u_{j+2}^* - 2u_{j+1}^* + 2u_{j-1}^* + u_{j-2}^*\right) + \gamma\frac{\Delta t}{2} I_j^n$$

где $r = \frac{\alpha\Delta t}{\Delta x2}$, $s = \frac{\beta\Delta t}{2\Delta x3}$

Приближение переносного члена в шаге «предиктор» для нечетных шагов по времени выполняется разностями вперед, а для четных шагов – разностями назад. С помощью этого ошибка фазы уменьшается в расчетах, что ведет за собой движение волновых фронтов в правильном направлении.

Известно, что схема Мак-Кормака порождает в окрестности сильных скачков колебания, носящие нефизический характер. При этом порядок аппроксимации повышается, по сравнению с монотонными схемами.

Алгоритм процедуры коррекции потоков в представлении [6] для координаты x состоит из следующих шагов:

1. Вычисление диффузионных потоков.

$$u^d_{j+\frac{1}{2}} = \nu_{j+\frac{1}{2}}(u^n_{j+1} - u^n_j).$$

2. Вычисление антидиффузионных потоков.

$$u^{ad}_{j+\frac{1}{2}} = \mu_{j+\frac{1}{2}}(\bar{u}_{j+1} - \bar{u}_j).$$

Коэффициенты $\nu_{j+\frac{1}{2}}$ и $\mu_{j+\frac{1}{2}}$ рассчитываются следующим способом:

$$\nu_{j+\frac{1}{2}} = \frac{1}{6} + \frac{1}{3}\mathbb{C}^2_{j+\frac{1}{2}}, \qquad \mu_{j+\frac{1}{2}} = \frac{1}{6} - \frac{1}{3}\mathbb{C}^2_{j+\frac{1}{2}}$$

$$\mathbb{C}^2_{j+\frac{1}{2}} = \frac{u^n_j + u^n_{j+1}}{2}\frac{\Delta t}{\Delta x}$$

После шага коррекции получаем отрегулированное значение u^{corr}_{jk}. Окончательное решение на шаге $n+1$ имеет следующий вид:

$$u^{n+1}_j = \bar{u}_j + u^d_{j+\frac{1}{2}} - u^d_{j-\frac{1}{2}} - u^{\text{corr}}_j + u^{\text{corr}}_{j-1}.$$

1.3 Уравнение Кадомцева-Петвиашвили

Не менее используемым в физических приложениях является многомерное обобщение уравнения Кортевега–де Вриза: уравнение Кадомцева–Петвиашвили.

Рассмотрим двухмерное уравнение:

$$[u_t + 0.5(u^2)_x + \beta u_{xxx} - G]_x = \eta u_{yy} \qquad (3)$$

Уравнение Кадомцева-Петвиашвили (3) относительно функции $u(x,y,t)$ обычно именуется уравнением КП и рассматривается в области при $t \geq 0$, $x, y \in (-\infty, \infty)$, $\beta, \eta \geq 0$, $G(x,y)$ — источник. В общем случае решение такого уравнения ищется численно [2].

1.3.1 Конечно-разностная схема для уравнения КП

При численном интегрировании уравнения КП вместо исходного уравнения (4) рассматривается его интегро-дифференциальный эквивалент:

$$u_t + 0.5(u^2)_x + \beta u_{xxx} = \eta \int_{-\infty}^{x} u_{yy}\,(x', y, t)dx' + G(x,y) \qquad (4)$$

Решение уравнения (5) в полуплоскости $t \geq 0$ ищется для начального распределения $u(x,y,0) = q(x,y)$.

В данной работе численное моделирование уравнения (4) проводится с помощью неявной линеаризованной конечно-разностной схемы с использованием в некоторых случаях процедуры коррекции потоков.

В данном случае непрерывная область для (4) заменяется на расчётную область $[x_{\min},\ x_{\max}] \times [y_{\min},\ y_{\max}] \times [0, T]$. В расчётной области задаётся равномерная разностная сетка по времени t и по пространственным координатам x и y:

$$x_j = j\Delta x, \qquad j \in [1, M], \qquad x_{\min} = x_1, \qquad x_{\max} = x_M$$

$$y_k = k\Delta y, \qquad k \in [1, L], \qquad y_{\min} = y_1, \qquad y_{\max} = y_L$$

$$t^n = n\Delta t, \qquad n = 0,1,2, \dots, T/\Delta t - 1$$

где $\Delta x, \Delta y$ — шаги по пространственным координатам, Δt — шаг по времени. В данных обозначениях для построенной сетки имеем:

$$u(j\Delta x, k\Delta y, n\Delta t) = u_{jk}^n, \quad \int_{-\infty}^{x_j} u_{yy}dx' \approx \int_{x_{\min}}^{x_j} u_{yy}dx' \equiv S_{jk}^n$$

Для уравнения (4) аппроксимация проводится с помощью центрально-разностных операторов:

$$u_{jk}^{n+1} - u_{jk}^{n} + \frac{\Delta t}{4\Delta x}\left(F_{j+1,k}^{n+1} - F_{j-1,k}^{n+1}\right) + \beta\frac{\Delta t}{2\Delta x^3}\left(u_{j+2,k}^{n+1} - 2u_{j+1,k}^{n+1} + 2u_{j-1,k}^{n+1} - u_{j-2,k}^{n+1}\right) = \Delta t\eta S_{jk}^{n+1} + \Delta t\eta G_{jk} \quad (5)$$

Порядок аппроксимации разностной схемы (5) в области расчёта составляет $O(\Delta t, \Delta x^2, \Delta y^2)$.

Получившаяся система разностных уравнений (5) приводится к виду:

$$a_j\Delta u_{j-2,k}^{n+1} + b_j\Delta u_{j-1,k}^{n+1} + c_j\Delta u_{jk}^{n+1} + d_j\Delta u_{j+1,k}^{n+1} + e_j\Delta u_{j+2,k}^{n+1} = f_{jk}^{n} \quad (6)$$

где $\Delta u_{jk}^{n+1} = u_{jk}^{n+1} - u_{jk}^{n}$.

При линеаризации конечно-разностного представления переносного члена для (6) имеем

$$F_{jk}^{n+1} \equiv (u^2)_{jk}^{n+1} = (u^2)_{jk}^{n} + 2u_{jk}^{n}\Delta u_{jk}^{n+1} + O(\Delta t^2)$$

Система (6) решается методом пятиточечной прогонки. [8]

Интеграл S_{jk}^{n} рассчитывается методом трапеций, подынтегральная производная u_{yy} аппроксимируется центральными разностями второго порядка:

$$\frac{\Delta t}{\Delta y^2}\left(u_{j,k-1}^{n+1} - 2u_{jk}^{n+1} + u_{j,k+1}^{n+1}\right).$$

1.3.2 Граничные условия

На границах расчётной области $[x_1, x_M] \times [y_1, y_L]$ задаются разностные граничные условия. Решения уравнения КП затухают пропорционально квадрату расстояния $O[1/(x^2 + y^2)]$, вследствие чего при практических расчётах нельзя выбрать достаточно большие размеры области $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ для использования однородных условий Дирихле. В таких случаях традиционно используют так называемые «условия протекания»: $u_x = u_{xx} = 0$ вдоль граничных линий x_1 и x_M, и $u_y = 0$ вдоль линий y_1 и y_L. В разностном виде дифференциальные условия выглядят так:

$$u^n_{-1,k} = u^n_{0,k} = u^n_{1,k};\ \ u^n_{M+2,k} = u^n_{M+1,k} = u^n_{M,k};\ \ u^n_{j,0} = u^n_{j,1};\ \ u^n_{j,L+1} = u^n_{j,L} \qquad (7)$$

где в системе (7) введены искусственные точки сетки:

$$(-1, k), (0, k), (M + 2, k), (M + 1, k), (j, 0), (j, L + 1).$$

Мы встречаемся здесь с особенностью применения разностных граничных условий (7). В действительности эти условия записаны для исходного дифференциального уравнения Кадомцева-Петвиашвили (3). При применении его интегро-дифференциального аналога (4) мы не получаем такого же результата для разностной формы (5). Это иллюстрация типичной проблемы, когда в непрерывном пространстве преобразования уравнения (3) к уравнению (4) тождественны, а в сеточном пространстве аналогичные преобразования из конечно-разностного представления уравнения (3) не приводят к выражению (5).

С учётом сказанного при расчётах системы уравнений (6) с граничными условиями (7) условия протекания по оси y не реализуются корректно, а приводят к условию отражения.

1.3.2 Начальные распределения

В качестве начальных распределений были выбраны три поверхности:

1. Параллелепипед.

Объём $V_1 = a_1 \times b_1 \times h$ где a_1, b_1, h — рёбра по осям x, y и z. При вычислении объёма в сеточной области учитываются значения шагов по пространственным координатам, так как практически рассматривается усечённая пирамида.

2. Распределение Гаусса.

$$q_2(x, y) = \sigma \exp[-\omega(x^2/a_2^2 + y^2/b_2^2)], \ \ \sigma > 0$$

Объём $V_2 = \sigma \pi a_2 b_2 / \omega$, где a_2, b_2 — полуоси эллипса.

3. Эллипсоид вращения.

$$\frac{x^2}{a_3^2} + \frac{y^2}{b_3^2} + \frac{z^2}{c_3^2} = 1, \ \ z \geq 0$$

Объём $V_3 = 2\pi a_3 b_3 / 3$, где a_3, b_3 — полуоси эллипса.

На Рис.1 представлены изображения начальных профилей 1-3. Высоты выбраны с условием, чтобы объём всех фигур был равным для сеточной области.

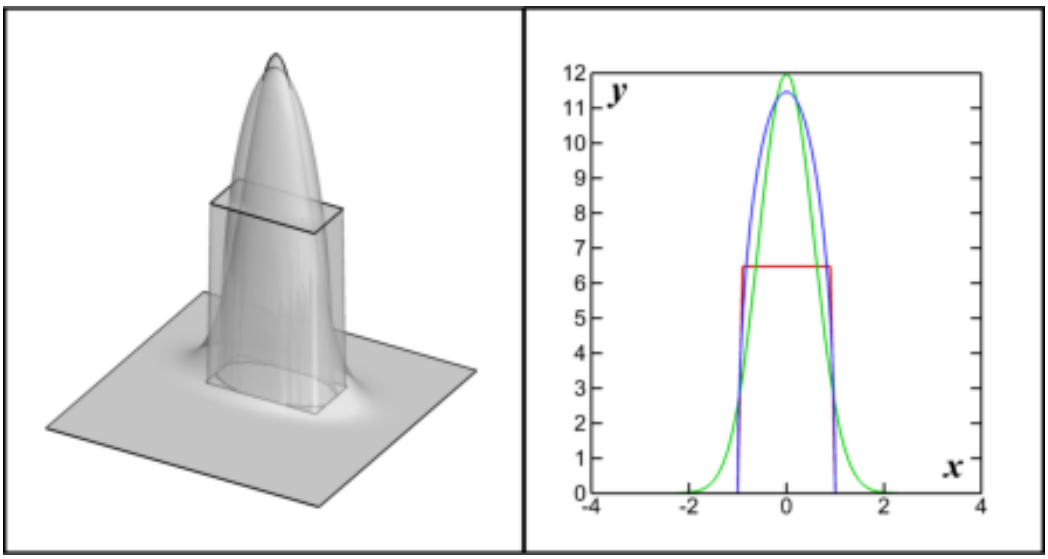

Рис. 1 – Начальные распределения

$$V = 48, a_1 = 2, b_1 = 4, a_2 = a_3 = 1, b_2 = b_3 = 2.$$

В большинстве случаев в качестве начального распределения выбирается аналитическое решение (3) при $G = 0$. В данном случае были выбраны распределение Гаусса с непрерывными производными; параллелепипед с разрывными производными (большими градиентами в сеточной области); эллипсоид вращения, у которого нет непрерывной производной при $z = 0$. Для унификации выбора значений параметров распределений мы фиксируем объём фигуры и выбор полуосей для эллипсов, которые вписываются в основание параллелепипеда.

Глава 2. Обзор архитектур

Графический процессор – Graphics Processing Unit (GPU), представляет собой специализированный микропроцессор, предназначенный для увеличения скорости вычислений. Использование графических процессоров распространено повсеместно.

Современные GPU очень эффективны, а их структура позволяет значительно превосходить классические центральные процессоры (CPU) по производительности. Графический процессор представляет собой многопоточную структуру с большим количеством ядер, что позволяет выполнять параллельную обработку информации, в отличие от CPU, который рассчитан на последовательные вычисления.

Многие приложения требуют большое количество массивных операций и расчетов, поэтому использование графического процессора делает решение задачи более действенным и дает точность на несколько порядков выше.

Все более распространенным становится использование графического процессора общего назначения (GPGPU). Первоначально GPU использовался только для ускорения компьютерной графики, а GPGPU – способ использования GPU для выполнения очень широкого класса вычислительных задач общего назначения, традиционно обрабатываемых центральным процессором (CPU).

Приведем некоторые примеры программных систем, предназначенных для параллельных вычислений.

- CUDA

Одна из основных компаний-производителей процессоров NVIDIA разработала платформу Compute Unified Device Architecture (CUDA), которая предоставляет разработчикам и инженерам прямой доступ к использованию

графического процессора с поддержкой CUDA для универсальных вычислений. Исследователи находят множество сфер применений CUDA. Например, с помощью графического процессора, который поддерживает данную вычислительную платформу, можно проводить исследования в области медицины или управлять воздушным движением [12].

CUDA имеет большое преимущество, которое заключается в программировании GPU на языках высокого уровня. Части программ работают на CPU, где выполняется последовательная часть кода и подготовительные стадии для вычислений на GPU . Другая часть программ выполняется на GPU и представляет собой параллельные участки кода, выполняемые одновременно при помощи специальных нитей (threads).

Для эффективного использования ресурсов GPU программе необходимо задействовать тысячи отдельных нитей, а на многоядерном CPU к максимальной эффективности можно прийти, если число потоков равно количеству ядер, или в несколько раз больше этого значения [8].

При работе с технологией CUDA также можно ознакомиться с полезными работами [9][11].

Однако для использования приведенной технологии от пользователя требуется определенный объем знаний, что затрудняет процесс программирования и решения задач.

- OpenCL

Также для эффективного решения прикладных задач используется распространенный открытый стандарт OpenCL. Эта технология может работать во множестве архитектур, в отличие от CUDA, которая предназначена для работы только с графическими процессорами NVIDIA.

OpenCL (Open Compute Language) – фреймворк для написания параллельных компьютерных программ, работающих на гетерогенных платформах, состоящих из различных процессоров. OpenCL значительно увеличивает скорость работы широкого спектра приложений, выполняющих высокопроизводительные вычисления для различных отраслей. В данный фреймворк входят язык программирования, основанный на стандарте C99 и интерфейс прикладного программирования (API). OpenCL предоставляет стандартный интерфейс для вычислений с использованием параллелизма на основе задач и данных [13].

- OpenMP

OpenMP (Open Multi-Processing) – это технология, предназначенная для распараллеливания программ на высокоуровневых языках программирования. OpenMP использует несколько ядер компьютера одновременно, тем самым увеличивая быстродействие при разработке приложений [14].

Важное достоинство технологии OpenMP: постепенное уменьшение нераспараллеленной части кода программы. Такой подход называется инкрементальным программированием и значительно увеличивает скорость компиляции программ.

Существует набор директив, вспомогательных функций и переменных окружения, с помощью которых последовательная часть программы изменяется на параллельную.

Программы с директивами OpenMP компилируются со специальным флагом – *fopenmp*. Параллельный код создается компилятором во время обработки директив.

Количество параллельных и последовательных областей не ограничено. Также важно заметить, что возможно вложение параллельных областей друг в друга [7].

Рассмотрим подробнее выполнение программы в параллельном режиме.

В языке C++ директивы оформляются в виде указаний, которые начинаются с *#pragma omp*. Здесь *omp* используется как флаг для того, чтобы можно было избежать повторения названий директив OpenMP и других имен программы.

Для запуска программы необходимо запустить следующую команду: *exsport OMP_NUM_THREADS = n*, которая задает количество потоков, выполняющих параллельную часть программы.

С помощью директивы *parallel [опция [[,] опция]…]* задается параллельная часть кода, которая может выполняться по условию, с предварительным заданием количества потоков, с заданием списка переменных, общим для всех количества потоков и с другими возможными опциями [7].

Для изменения последовательной версии программы требуется сначала найти части кода, которые могут выполняться независимо разными потоками. В большинстве случаев эффективное создание параллельного варианта программы выполняется с помощью распараллеливания циклов.

Главное преимущество технологии OpenMP заключается в том, что созданная параллельная версия программы может использоваться и как последовательная. В этом случае директивы OpenMP просто будут проигнорированы компилятором.

Описанная выше технология может использоваться совместно с другими программными интерфейсами, что существенно увеличивает быстродействие программы и делает параллельную обработку более результативной.

Глава 3. Разработка алгоритма

В данной главе будет рассмотрено построение алгоритма решения неоднородного обобщённого двухмерного уравнения Кадомцева-Петвиашвили:

$$[u_t + 0.5(u^2)_x + \beta u_{xxx} - G]_x = \eta u_{yy}$$

Алгоритм реализован на языке программирования C++. Для распараллеливания использовался открытый стандарт OpenMP.

Введены следующие обозначения:

- *u* – массив функции на слое n, размерности *j2xk2*;
- *i_shape* – начальное рапределение;
- *u0_max* – высота параллепипеда;
- *ta* – шаг по времени;
- *dx, dy* – шаги по координатам;
- *t_stop* – максимальное время вычисления;
- *istep* – количество шагов.

Во время реализации алгоритма для хранения результатов и некоторых промежуточных значений используются массивы:

- *ur* – полусумма скоростей;
- *cu_d* – диффузионные коэффициенты;
- *cu_ad* – антидиффузионные коэффициенты;
- *u_d* – значения диффузионого потока на этапе «коррекции потоков»;
- *u_ad* –значения антидиффузионого потока на этапе «коррекции потоков»;
- *d_u* – массив первых разностей.

#pragma omp parallel for private(...) - директива OpenMP, использующаяся для задания параллельной области;

Данная директива применяется для распараллеливания циклов, выполняющих массовые расчеты, с целью ускорения процесса решения.

Часть кода программы, реализующая метод пятиточечной прогонки, приведена в Приложении 1.

Зафиксированы значения параметров:

$a_1 = 4, \quad b_1 = 6, \quad a_2 = a_3 = 2, \quad b_2 = b_3 = 3, \quad V = 120.$

В данном случае расчет проводился для момента времени: $t\_stop = 8$.

Шаги по координатам: $dx = dy = 0.1$.

Шаг по времени: $ta = 5 * 10^{-5}$.

На Рис. 2 представлен результат расчета, зависящий от начального распределения.

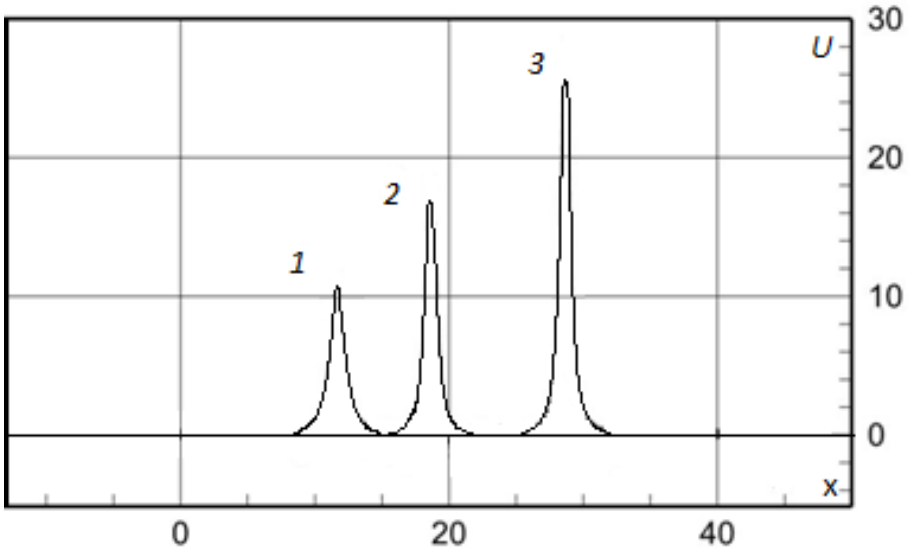

Рис.2 – Результаты расчета

Как очевидно вытекает из Рис. 2: наибольший солитон (уединенная волна) образуется из первоначальной формы эллипсоида вращения.

Также были проведены исследования высоты солитона от первоначального объема.

Расчеты проводились для начального распределения гауссовского типа для следующих параметров:

$$a_2 = 2, \qquad b_2 = 3, \qquad ta = 10^{-4}, \qquad dx = dy = 0.2, \qquad t\_stop = 7.$$

Таблица 2. Зависимость высоты от начального объема

№	V	σ
1	96	8
2	108	9
3	120	10
4	132	11
5	138	11,5
6	141	11,75
7	142,5	11,875
8	143,4	11,95
9	143,64	11,97
10	144	12

На Рис. 3 проиллюстрировано увеличение амплитуды солитона для начальных условий распределения Гаусса.

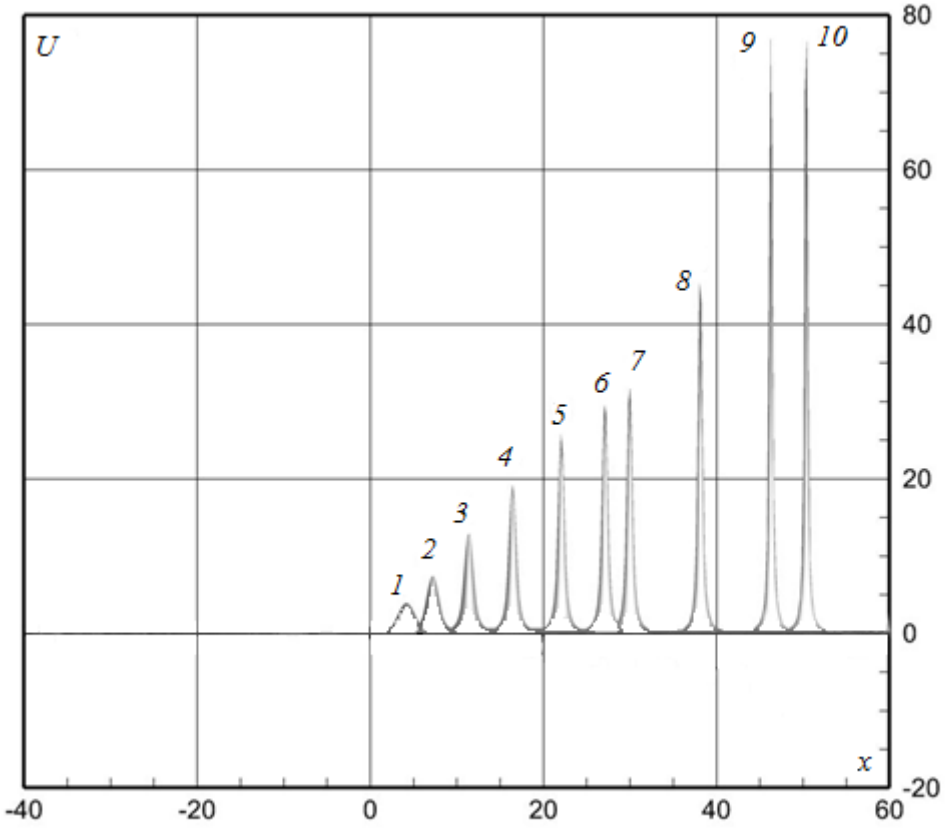

Рис.3 – Увеличение амплитуды солитона

Чтобы увеличить скорость быстродействия программы необходимо использовать все доступные ядра вычислительной платформы, если платформа является многоядерной.

Для наглядности программа, реализующая распараллеливание данного алгоритма, была запущена на многоядерной машине на различных ядрах, в результате чего были получены ускорения, которые приведены в Таблице 1.

Таблица 1 - Результат работы программы

Количество ядер	Ускорение
1	1
3	1.4
6	2.8
12	5.7

Выводы

В данной работе была поставлена задача разработать параллельный алгоритм для сильно нелинейных волновых процессов. Для этого были изучены основные сведения об уравнениях, описывающих нелинейные явления. В большинстве случаев, решение таких дифференциальных уравнений предполагает огромное количество массовых расчетов, что значительно замедляет процесс визуализации. Поэтому были проанализированы технологии, с помощью которых можно добиться увеличения скорости работы алгоритма. В качестве оптимального решения для распараллеливания программного кода был выбран открытый стандарт OpenMP.

Реализованная программа основана на методе пятиточечной прогонки и решает двухмерное уравнение Кадомцева–Петвиашвили. Технология OpenMP используется для осуществления параллельных вычислений.

Кроме того, был проведен запуск данного алгоритма на вычислительной платформе с использованием различного количества ядер. Данный эксперимент наглядно показал, как ускоряется работа программы при увеличении количества ядер вычислительной машины.

Поставленная задача была решена в полной мере.

Заключение

Решение нелинейных уравнений представляет собой очень трудоемкий процесс, и производительности универсальных CPU не всегда хватает для массовых расчетов в реальном времени. Поэтому использование различных архитектур для ускорения вычислений является популярной темой на сегодняшний день.

В результате данной работы разработан и распараллелен алгоритм решения дифференциальных уравнений, описывающих нелинейные процессы.

Освоена работа с технологией OpenMP, позволяющей значительно ускорить процесс нахождения решения дифференциального уравнения.

Список литературы

1. Bogdanov A.V., Mareev V.V. Numerical Simulation of the Perturbed KdVB Equation //EPJ Web of Conferences. – EDP Sciences, 2016. – T.108. – C.02014.
2. Bogdanov A.V., Mareev V.V. Numerical Simulation KPI Equation //15th International Ship Stability Workshop, Stockholm, Sweden. – 2016.
3. Boris J. P., Book D. L. Flux-corrected transport. I. SHASTA, A fluid transport algorithm that works //Journal of computational physics. – 1973. –T. 11. –No. 1. –C. 38-69.
4. Zalesak S. T. Fully multidimensional flux-corrected transport algorithms for fluids //Journal of computational physics. –1979. –T. 31. –No. 3. – C. 335-362.
5. Fletcher R. H., Tannehill J. C., and Anderson D. Computational Fluid Mechanics and Heat Transfer, Third Edition. – CRC Press, 2013, 774 p.
6. Fletcher C. A. J., Computational Techniques for Fluid Dynamics 2: Specific Techniques for Different Flow Categories. – (Springer-Verlag, 1998), 496 p.
7. Антонов А.С. Параллельное программирование с использованием технологии OpenMp. – М.: Изд-во МГУ, 2009. – 77 р.
8. Боресков А. В. и др. Параллельные вычисления на GPU //Архитектура и программная модель CUDA. М.: Изд-во Московского университета. – 2012.
9. Боресков А., Харламов А. Основы работы с технологией CUDA. –Litres, 2015.
10. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений //Метод пятиточечной прогонки. – 1978, 592 р.

11. Сандерс Д., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров //М.: ДМК Пресс. – 2011. – Т. 232.
12. CUDA. http://www.nvidia.com/object/cuda_home_new.html
13. OpenCL. https://www.khronos.org/registry/OpenCL/specs/opencl-2.2.pdf
14. OpenMP. http://www.openmp.org/

Приложение 1.

Реализация метода пятиточечной прогонки:

```
for (k = 1; k <= k2; k++)
     {
          kh=k-1;
          kb=k+1;
          cx_1=1.0+caa-cdx7*u0[1][k];
          dx1=cdx7*u0[2][k]-cb;
          ce=caa;

          fx1=ta*(S_eta[1][k]+S_gam[1][k]);
          fx1=fx1-cdx1*(u0[2][k]*u0[2][k]-u0[1][k]*u0[1][k]);
          fx1=fx1-caa*(u0[3][k]-2.0*u0[2][k]+u0[1][k]);

          alfa[2]=-dx1/cx_1;
          beta[2]=ce/cx_1;
          gamma[2]=fx1/cx_1;
          bx2=caa-cdx7*u0[1][k];
          cc=1.0;
          dx2=cdx7*u0[3][k]-cb;
          ce=caa;
          fx2=ta*(S_eta[2][k]+S_gam[2][k]);
          fx2=fx2-cdx1*(u0[3][k]*u0[3][k]-u0[1][k]*u0[1][k]);
          fx2=fx2-caa*(u0[4][k]-2.0*u0[3][k]+u0[1][k]);

          ab=1.0/(cc+alfa[2]*bx2);
          alfa[3]=ab*(-dx2+beta[2]*bx2);
          beta[3]=ab*ce;
          gamma[3]=ab*(fx2-gamma[2]*bx2);

          for (j = 3; j <= j7; j++)
          {
```

```
                jl=j-1;
                jl2=j-2;
                jr=j+1;
                jr2=j+2;
                ca=-caa;
                bxj=cb-cdx7*u0[jl][k];
                cc=1.0;
                dxj=-cb+cdx7*u0[jr][k];
                ce=caa;

                fxj=ta*(S_eta[j][k]+S_gam[j][k]);
                fxj=fxj-cdx1*(u0[jr][k]*u0[jr][k]-
u0[jl][k]*u0[jl][k]);
                fxj=fxj-caa*(u0[jr2][k]-
2.0*u0[jr][k]+2.0*u0[jl][k]-u0[jl2][k]);

                ab=1.0/(cc-beta[jl]*ca+alfa[j]*(alfa[jl]*ca+bxj));
                alfa[jr]=ab*(-dxj+beta[j]*(alfa[jl]*ca+bxj));
                beta[jr]=ab*ce;
                gamma[jr]=ab*(fxj-gamma[jl]*ca-
gamma[j]*(alfa[jl]*ca+bxj));
          }

          ca=-caa;
          bxj9=cb-cdx7*u0[j7][k];
          cc=1.0;
          dxj9=cdx7*u0[j2][k]-caa;

          fxj9=ta*(S_eta[j9][k]+S_gam[j9][k]);
          fxj9=fxj9-cdx1*(u0[j2][k]*u0[j2][k]-
u0[j7][k]*u0[j7][k]);
          fxj9=fxj9-caa*(-u0[j2][k]+2.0*u0[j7][k]-u0[j9-2][k]);

          ab=1.0/(cc-beta[j7]*ca+alfa[j9]*(alfa[j7]*ca+bxj9));
```

```
          gamma[j2]=ab*(fxj9-gamma[j7]*ca-
gamma[j9]*(alfa[j7]*ca+bxj9));

          ca=-caa;
          bxj2=cb-cdx7*u0[j9][k];
          cx_j2=1.0-caa+cdx7*u0[j2][k];

          fxj2=ta*(S_eta[j2][k]+S_gam[j2][k]);
          fxj2=fxj2-cdx1*(u0[j2][k]*u0[j2][k]-
u0[j9][k]*u0[j9][k]);
          fxj2=fxj2-caa*(-u0[j2][k]+2.0*u0[j9][k]-u0[j7][k]);

          ab=1.0/(cx_j2-beta[j9]*ca+alfa[j2]*(alfa[j9]*ca+bxj2));
          gamma[j2+1]=ab*(fxj2-gamma[j9]*ca-
gamma[j2]*(alfa[j9]*ca+bxj2));

          u1[j2][k]=gamma[j2+1];
          u1[j9][k]=alfa[j2]*u1[j2][k]+gamma[j2];

          for (j = j7; j >= 1; j--)
          {
               jr=j+1;
               u1[j][k]=alfa[jr]*u1[jr][k]-
beta[jr]*u1[j+2][k]+gamma[jr];
          }
     }
```